On Linear Logic, Functional Programming, and Attack Trees

Harley Eades III[1], Jiaming Jiang[2], and Aubrey Bryant[1]

[1] Computer Science, Augusta University, harley.eades@gmail.com
[2] Computer Science, North Carolina State University

Abstract. TODO

1 Introduction

Attack trees are perhaps the most popular graphical model used to conduct threat analysis of both physical and virtual secure systems. They were made popular by Bruce Schneier in the late nineties [17]. In those early years attack trees were studied and used as a syntactic tool to help guide analysis. However, as systems grew more complex the need for a semantics of attack trees become apparent, after all, without a proper semantics how can we safely manipulate attack trees, extend the expressivity of attack trees, or compare them?

A number of different models of attack trees have been proposed: a model in boolean algebras [12,11,16], series-parallel pomsets [13], Petri nets [14], and tree automata [1]. There have also been various extensions, such as, adding sequential composition [7], and defense nodes [10,11]. All of these models and extensions have their benefits, but at the heart of them all is logic.

The model in boolean algebras was the first and most elegant model of attack trees, but it failed to capture the process notion of attack trees, that is, the fact that base attacks are actual processes that need to be carried out, and the branching nodes composed these processes in different ways. Thus, the community moved towards models of resources like parallel-series pomsets, Petri nets, and automata. However, the complexity of these models increased, and hence, comparing the models is difficult which makes it hard to decide which to use and under which circumstances, and so we have wondered, is there a means of recovering the elegant model in logic, and can this logic teach us anything new?

Linear Logic. It is fitting that attack trees are the most popular model used in threat analysis, because one of the most widely studied logics used to reason about resources is *linear logic* which is an excellent candidate for modeling attack trees. In fact, Horne et al.[6] has already produced a number of interesting results. Most importantly, they show that attack trees can be modeled as formulas in linear logic, and then one can prove properties between attack trees by proving implications between them. Furthermore, by studying attack trees from a linear logical perspective they introduce a new property between attack trees called *specializations*. Prior to their paper the literature was primarily concerned with equality between attack trees, but implication allows us to break that equality

up. A specialization is a one way relationship between two attack trees that may even be influenced by the attack trees attribute domain; we will discuss specializations in more detail in Section **??**.

This paper has two main contributions, the first is a new simple linear logical semantics of causal attack trees – attack trees with sequential composition – in four-valued truth tables. We show that our semantics is surprisingly expressive. It supports specializations, and even lays outside of the semantics proposed by Horne et al.[6], because it combines in an interesting way what they call the *ideal* and *filter* semantics of causal attack trees. However, the most appealing aspect of this semantics is that it is extremely simple. Furthermore, we introduce a family of natural deduction systems based in the logic of bunched implications which were formed by studying the truth table semantics.

Functional Programming. Our second contribution is Lina, a new domain specific functional programming language for conducting threat analysis using attack trees. Consider the example attack trees in Fig. 1. Both of these are

A.

```
seq_node "ATM attack"
 (and_node "get credientials"
   (base_na "steal card")
   (or_node "get PIN"
    (base_na "social engineer")
    (base_na "find a post-it")))
 (base_na "withdraw money")
```

B.

```
or_node "ATM attack"
 (seq_node "attack vector 1"
    (and_node "get credentials 1"
       (base_na "social engineer")
       (base_na "steal card"))
    (base_na "withdraw money"))
 (seq_node "attack vector 2"
    (and_node "get credentials 2"
       (base_na "steal card")
       (base_na "find a post-it"))
    (base_na "withdraw money"))
```

Fig. 1. Attack Tree for an ATM attack from Figure 1 (A) and Figure 2 (B) of Kordy et al. [9]

actual Lina programs, in fact every example in this paper are Lina programs. Lina supports causal attack trees both with attributes or without, thus, there are two types of base attacks: base attacks with attributes, denoted `base_wa`, and base attacks with no attributes, denoted `base_na`. Lina is designed to be extremely simple, and actually reflect the typical pseudocode found throughout the literature. However, Lina is more than just a simple definitional language.

Lina is an embedded domain specific language whose host language is the Haskell programming language [8]. So, why purely functional and why strongly typed? As security researchers/professionals we are in the business of verifying the correctness of various systems. Thus, our tools should be taking advantage of verification tools to insure that our constructions, tools, and analysis are correct. By embedding Lina into Haskell we are able to take advantage of cutting edge verification tools while conducting threat analysis. For example, right out the box

Lina supports using property-based randomized testing using QuickCheck [2], and refinement types in Liquid Haskell [18] to verify properties of our attack trees or the attribute domains used while analyzing attack trees. Furthermore, strong typing helps catch bugs while we develop our attack trees and their attribute domains as a side-effect of type checking. Finally, functional programs are short, but not obfuscated, and hence, allow for very compact and trustworthy programs.

That being said, we are designing Lina so that it can be used with very little Haskell experience. It is our hope that one will be able to make use of Lina without having to know how to write Haskell programs, and we plan to develop new tooling to support this.

Lina is approaching threat analysis from a programming language perspective. This approach leads to a number of new advances. First, as Gadyatskaya and Trujillo-Rasua [5] argue as a community we need to start building more automated means of conducting threat analysis, and there is no better way to build or connect automated tools than a programming language. Lina is perfect as a target for new tools, and it can be connected to existing tools fairly easily. In fact, Lina already supports automation using the automatic rewrite system Maude [3], for example, the two attack trees in Fig. 1 can be automatically proven equivalent to each other in Lina. This is similar to Krody's [9] SPTool, but Lina goes further and supports more than one backend rewrite system, for example, Lina is the first tool to support automatically proving specializations of attack trees. The user can choose which backend they wish to use.

We have a number of extensions planned for the future like supporting attack-defense trees, attack(-defense) graphs, and attack nets. We plan to support even more automation using SAT and SMT solvers. It is our hope that Lina grows into a one stop shop for threat analysis.

2 SAND Attack Trees

In this section we introduce SAND attack trees. This formulation of attack trees was first proposed by Jhawar et al. [7].

Definition 1. *Suppose* B *is a set of base attacks whose elements are denoted by* b*. Then an* ***attack tree*** *is defined by the following grammar:*

$$A, B, C, T := b \mid \mathsf{OR}(A, B) \mid \mathsf{AND}(A, B) \mid \mathsf{SAND}(A, B)$$

Equivalence of attack trees, denoted by $A \approx B$*, is defined as follows:*

$$
\begin{array}{ll}
(E_1) & \mathsf{OR}(\mathsf{OR}(A, B), C) \approx \mathsf{OR}(A, \mathsf{OR}(B, C)) \\
(E_2) & \mathsf{AND}(\mathsf{AND}(A, B), C) \approx \mathsf{AND}(A, \mathsf{AND}(B, C)) \\
(E_3) & \mathsf{SAND}(\mathsf{SAND}(A, B), C) \approx \mathsf{SAND}(A, \mathsf{SAND}(B, C)) \\
(E_4) & \mathsf{OR}(A, B) \approx \mathsf{OR}(B, A) \\
(E_5) & \mathsf{AND}(A, B) \approx \mathsf{AND}(B, A) \\
(E_6) & \mathsf{AND}(A, \mathsf{OR}(B, C)) \approx \mathsf{OR}(\mathsf{AND}(A, B), \mathsf{AND}(A, C)) \\
(E_7) & \mathsf{SAND}(A, \mathsf{OR}(B, C)) \approx \mathsf{OR}(\mathsf{SAND}(A, B), \mathsf{SAND}(A, C))
\end{array}
$$

A.

```
and_node "obtain secret"
 (or_node "obtain encryped file"
    (base_na "bribe sysadmin")
    (base_na "steal backup"))
 (seq_node "obtain password"
    (base_na "break into system")
    (base_na "install keylogger"))
```

B.

```
and_node "obtain secret"
 (or_node "obtain encryped file"
    (base_na "bribe sysadmin")
    (base_na "steal backup"))
 (seq_node "obtain password"
    (base_na "break into system")
    (base_na "install keylogger"))
```

C.

```
or_node "obtain secret"
 (and_node "obtain secret via sysadmin"
  (base_na "bribe sysadmin")
  (seq_node "obtain password"
    (base_na "break into system")
    (base_na "install keylogger")))
  (seq_node "break in, then obtain secrect"
    (base_na "break into system")
    (and_node "obtain secret from inside"
      (base_na "install keylogger")
      (base_na "steal backup")))
```

Fig. 2. Encrypted Data Attack from Figure 1 (A), Figure 3 (B), and Figure 2 (C) of Horne et al. [6]

```
vehicle_attack :: APAttackTree Double String
vehicle_attack = start_PAT $
  or_node "Autonomous Vehicle Attack"
    (seq_node "external sensor attack"
       (base_wa 0.2 "modify street signs to cause wreck")
       (and_node "social engineering attack"
          (base_wa 0.6 "pose as mechanic")
          (base_wa 0.1 "install malware")))
    (seq_node "over night attack"
       (base_wa 0.05 "Find address where car is stored")
       (seq_node "compromise vehicle"
          (or_node "break in"
             (base_wa 0.8 "break window")
             (base_wa 0.5 "disable door alarm/locks"))
          (base_wa 0.1 "install malware")))
```

Fig. 3. Lina Script for an Autonomous Vehicle Attack

```
OR("Autonomous Vehicle Attack",0.6)
  (SEQ("external sensor attack",0.6)
     ("modify street signs to cause wreck",0.2)
     (AND("social engineering attack",0.6)
        ("pose as mechanic",0.6)
        ("install malware",0.1)))
```


```
OR("Autonomous Vehicle Attack",0.6)
  (SEQ("external sensor attack",0.6)
     ("modify street signs to cause wreck",0.2)
     (AND("social engineering attack",0.6)
        ("pose as mechanic",0.6)
        ("install malware",0.1)))
```

```
OR("Autonomous Vehicle Attack",0.6)
  (SEQ("external sensor attack",0.6)
     ("modify street signs to cause wreck",0.2)
     (AND("social engineering attack",0.6)
        ("pose as mechanic",0.6)
        ("install malware",0.1)))
```


Fig. 4. Set of Possible Attacks for an Autonomous Vehicle Attack

This definition of SAND attack trees differs slightly from Jhawar et. al.'s [7] definition. They define n-ary operators, but we only consider the binary case, because it fits better with the models presented here and it does not loose any generality because we can model the n-ary case using binary operators in the obvious way. Finally, they also include the equivalence $\mathsf{OR}(A, A) \approx A$, but it is not obvious how to include this in the models presented here and we leave its addition to future work.

3 A Quaternary Semantics for SAND Attack Trees

Kordy et al. [11] gave a very elegant and simple semantics of attack-defense trees in boolean algebras. Unfortunately, while their semantics is elegant it does not capture the resource aspect of attack trees, it allows contraction, and it does not provide a means to model sequential conjunction. In this section we give a semantics of attack trees in the spirit of Kordy et al.'s using a four valued logic.

The propositional variables of our quaternary logic, denoted by A, B, C, and D, range over the set $\mathsf{4} = \{0, \frac{1}{4}, \frac{1}{2}, 1\}$. We think of 0 and 1 as we usually do in boolean algebras, but we think of $\frac{1}{4}$ and $\frac{1}{2}$ as intermediate values that can be used to break various structural rules. In particular we will use these values to prevent exchange for sequential conjunction from holding, and contraction from holding for parallel and sequential conjunction.

Definition 2. *The logical connectives of our four valued logic are defined as follows:*

Parallel and Sequential Conjunction:

$A \odot_4 B = 1$,
where neither A *nor* B *are* 0
$A \odot_4 B = 0$, *otherwise*

$A \rhd_4 B = 1$,
where $A \in \{\frac{1}{2}, 1\}$ *and* $B \neq 0$
$\frac{1}{4} \rhd_4 B = \frac{1}{4}$, *where* $B \neq 0$
$A \rhd_4 B = 0$, *otherwise*

Choice: $A \sqcup_4 B = \mathsf{max}(A, B)$

These definitions are carefully crafted to satisfy the necessary properties to model attack trees. Comparing these definitions with Kordy et al.'s [11] work we can see that choice is defined similarly, but parallel conjunction is not a product – ordinary conjunction – but rather a linear tensor product, and sequential conjunction is not actually definable in a boolean algebra, and hence, makes heavy use of the intermediate values to insure that neither exchange nor contraction hold.

We use the usual notion of equivalence between propositions, that is, propositions ϕ and ψ are considered equivalent, denoted by $\phi \equiv \psi$, if and only if they have the same truth tables. In order to model attack trees the previously defined logical connectives must satisfy the appropriate equivalences corresponding to the equations between attack trees. These equivalences are all proven by the following result.

Lemma 1 (Properties of the Attack Tree Operators in the Quaternary Semantics).

(Symmetry) For any A *and* B, $A \bullet B \equiv B \bullet A$, *for* $\bullet \in \{\odot_4, \sqcup_4\}$.

(Symmetry for Sequential Conjunction) It is not the case that, for any A *and* B, $A \rhd_4 B \equiv B \rhd_4 A$.

(Associativity) For any A, B, *and* C, $(A \bullet B) \bullet C \equiv A \bullet (B \bullet C)$, *for* $\bullet \in \{\odot_4, \rhd_4, \sqcup_4\}$.

(Contraction for Parallel and Sequential Conjunction) It is not the case that for any A, $A \bullet A \equiv A$, *for* $\bullet \in \{\odot_4, \rhd_4\}$.

(Distributive Law) For any A, B, *and* C, $A \bullet (B \sqcup_4 C) \equiv (A \bullet B) \sqcup_4 (A \bullet C)$, *for* $\bullet \in \{\odot_4, \rhd_4\}$.

Proof. Symmetry, associativity, contraction for choice, and the distributive law for each operator hold by simply comparing truth tables. As for contraction for parallel conjunction, suppose $A = \frac{1}{4}$. Then by definition $A \odot_4 A = 1$, but $\frac{1}{4}$ is not 1. Contraction for sequential conjunction also fails, suppose $A = \frac{1}{2}$. Then by definition $A \rhd_4 A = 1$, but $\frac{1}{2}$ is not 1. Similarly, symmetry fails for sequential conjunction. Suppose $A = \frac{1}{4}$ and $B = \frac{1}{2}$. Then $A \rhd_4 B = \frac{1}{4}$, but $B \rhd_4 A = 1$.

At this point it is quite easy to model attack trees as formulas. The following defines their interpretation.

Definition 3. *Suppose* $\mathbb{B}$ *is some set of base attacks, and* $\nu : \mathbb{B} \to \mathsf{PVar}$ *is an assignment of base attacks to propositional variables. Then we define the interpretation of* ATerms *to propositions as follows:*

$$\begin{array}{lcllcl} [\![b \in \mathbb{B}]\!] & = & \nu(b) & [\![\mathsf{SAND}(A,B)]\!] & = & [\![A]\!] \rhd_4 [\![B]\!] \\ [\![\mathsf{AND}(A,B)]\!] & = & [\![A]\!] \odot_4 [\![B]\!] & [\![\mathsf{OR}(A,B)]\!] & = & [\![A]\!] \sqcup_4 [\![B]\!] \end{array}$$

We can use this semantics to prove equivalences between attack trees.

Lemma 2 (Equivalence of Attack Trees in the Quaternary Semantics). *Suppose* $\mathbb{B}$ *is some set of base attacks, and* $\nu : \mathbb{B} \to \mathsf{PVar}$ *is an assignment of base attacks to propositional variables. Then for any attack trees* A *and* B*, if* $A \approx B$*,* $[\![A]\!] \equiv [\![B]\!]$*.*

Proof. This proof holds by induction on the form of $A \approx B$.

This is a very simple and elegant semantics, but it also leads to a more substantial theory.

4 The Attack Tree Linear Logic (ATLL)

In this section we take what we have learned by constructing the dialectica model and define a intuitionistic linear logic, called the attack tree linear logic (ATLL), that can be used to prove equivalences between attack trees as linear implications. ATLL is based on the logic of bunched implications (BI) [15], in that, contexts will be trees. This is necessary to be able to include parallel and sequential conjunction, and choice within the same logic, because they all have different structural rules associated with them.

The syntax for formulas and contexts are defined by the following grammar.

$$\begin{array}{l} A, B, C, D, T := N \mid A \sqcup B \mid A \odot B \mid A \rhd B \mid A \multimap B \\ \Gamma, \Delta := * \mid A \mid \Gamma, \Delta \mid \Gamma; \Delta \mid \Gamma \bullet \Delta \end{array}$$

ATLL formulas are not surprising, but we denote base attacks by atomic formulas represented here by N. The syntax for contexts are similar to the contexts in BI. Contexts are trees with three types of nodes denoted by Γ, Δ for parallel conjunction, $\Gamma; \Delta$ for sequential conjunction, and $\Gamma \bullet \Delta$ for choice. They all have units, but we overload the symbol $*$ to represent them all.

The ATLL inference rules are given in Figure 5. The inference rules are fairly straightforward. We denote by $\Delta(\Gamma)$ the context Δ with a subtree – subcontext – Γ. This syntax is used to modify the context across inference rules.

Perhaps the most interesting rule is the CM rule which stands for context morphism. This rule is a conversion rule for manipulation of the context. It depends on a judgment $\Gamma_1 \vdash \Gamma_2$ which can be read as the context Γ_1 can be transformed into the context Γ_2. This judgment is defined by the rules in Figure 6. Context morphisms are designed to induce structural rules for some of the logical connectives and not for others. For example, parallel conjunction and choice should be commutative, but sequential conjunction should not be. The

$$\frac{}{B \vdash B}\ \text{L\_VAR} \qquad \frac{}{* \vdash N}\ \text{L\_NODE} \qquad \frac{\Gamma_1 \vdash \Gamma_2 \quad \Gamma_2 \vdash A}{\Gamma_1 \vdash A}\ \text{L\_CTX}$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \bullet \Delta \vdash A \odot B}\ \text{L\_PARAI} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma \blacksquare \Delta \vdash A \sqcup B}\ \text{L\_CHOICEI}$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma; \Delta \vdash A \rhd B}\ \text{L\_SEQI} \qquad \frac{\Gamma \vdash A \odot B \quad \Delta(A \bullet B) \vdash C}{\Delta(\Gamma) \vdash C}\ \text{L\_PARAE}$$

$$\frac{\Gamma \vdash A \sqcup B \quad \Delta(A \blacksquare B) \vdash C}{\Delta(\Gamma) \vdash C}\ \text{L\_CHOICEE}$$

$$\frac{\Gamma \vdash A \rhd B \quad \Delta(A; B) \vdash C}{\Delta(\Gamma) \vdash C}\ \text{L\_SEQE} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ \text{L\_LIMPI}$$

$$\frac{\Gamma \vdash A \multimap B \quad \Delta \vdash A}{\Gamma, \Delta \vdash B}\ \text{L\_LIMPE}$$

Fig. 5. ATLL Inference Rules

$$\frac{}{\Gamma \vdash \Gamma}\ \text{C\_ID} \qquad \frac{\Gamma_1 \vdash \Gamma_2 \quad \Gamma_2 \vdash \Gamma_3}{\Gamma_1 \vdash \Gamma_3}\ \text{C\_C}$$

$$\frac{}{(\Gamma_1 \circ \Gamma_2) \circ \Gamma_3 \vdash \Gamma_1 \circ (\Gamma_2 \circ \Gamma_3)}\ \text{C\_A1} \qquad \frac{}{\Gamma \circ * \vdash \Gamma}\ \text{C\_U1}$$

$$\frac{}{* \circ \Gamma \vdash \Gamma}\ \text{C\_U2} \qquad \frac{}{\Gamma(A, B) \vdash \Gamma(B, A)}\ \text{C\_E1}$$

$$\frac{}{\Gamma(A \bullet B) \vdash \Gamma(B \bullet A)}\ \text{C\_E2}$$

$$\frac{}{\Gamma(A; (\Delta_1 \bullet \Delta_2)) \vdash \Gamma((A; \Delta_1) \bullet (A; \Delta_2))}\ \text{C\_D1}$$

$$\frac{}{\Gamma((A; \Delta_1) \bullet (A; \Delta_2)) \vdash \Gamma(A; (\Delta_1 \bullet \Delta_2))}\ \text{C\_D2}$$

$$\frac{}{\Gamma(A, (\Delta_1 \bullet \Delta_2)) \vdash \Gamma((A, \Delta_1) \bullet (A, \Delta_2))}\ \text{C\_D3}$$

$$\frac{}{\Gamma((A, \Delta_1) \bullet (A, \Delta_2)) \vdash \Gamma(A, (\Delta_1 \bullet \Delta_2))}\ \text{C\_D4}$$

Fig. 6. Context Morphisms

rules for associativity and the unit rules mention the operator, $\circ$, this operator ranges over ',', ';', and '$\bullet$'.

One interesting, and novel aspect of this logic in contrast to BI is we can use the context morphisms to induce distributive laws between the various tensor products. The rules dist_1 and dist_2 induce the property that sequential conjunction distributes over choice, and dist_3 and dist_4 induce the property that parallel conjunction distributes over choice.

The interpretation of attack trees as ATLL formulas is obvious at this point where base attacks are atomic formulas and we denote this interpretation as $[\![T]\!]$ for some attack tree T. The most interesting part about this interpretation is that we can now use linear implication to prove properties about attack trees. First, we can derive all of the required equivalences in ATLL.

Lemma 3 (Attack Tree Logical Equivalences). *The following hold for any ATLL formulas A, B, and C.*

- $* \vdash (A \sqcup B) \multimap (B \sqcup A)$
- $* \vdash (A \odot B) \multimap (B \odot A)$
- $* \vdash ((A \sqcup B) \sqcup C) \multimap (A \sqcup (B \sqcup C))$
- $* \vdash ((A \odot B) \odot C) \multimap (A \odot (B \odot C))$
- $* \vdash ((A \rhd B) \rhd C) \multimap (A \rhd (B \rhd C))$
- $* \vdash (A \odot (B \sqcup C)) \multimap ((A \odot B) \sqcup (A \odot C))$
- $* \vdash (A \rhd (B \sqcup C)) \multimap ((A \rhd B) \sqcup (A \rhd C))$

Using the previous lemma we can now completely reason about equivalences of attack trees in ATLL. Another important aspect of ATLL is that we can use either the left-to-right directions or the right-to-left directions of the previous bi-implications to simplify attack trees into normal forms. In addition, the logical interpretation leads to new and interesting questions, for example, adding additional structural rules, like weakening, could also open the door for proving when one attack tree is a subattack tree of another. This concept is yet to appear in the literature, but has practical applications.

5 Related and Future Work

Related Work. Horne et al. [6] also propose modeling SAND attack trees using linear logic, but they base their work on pomsets and classical linear logic. In addition, their logic cannot derive the distributive law for sequential conjunction up to an equivalence, but they can derive `<<no parses (char 2): * ***|- ((A > B) + (A > C)) -o (A > (B` The full equivalence is derivable in ATLL however.

The logic of bunched implications [15] has already been shown to be able to support non-commutative operators by O'hearn, but here we show how distributive laws can be controlled using properties on contexts.

de Paiva [4] shows how to model non-commutative operators in dialectica categories, but here we show an alternative way of doing this, and we extend the model to include more operators like choice and its distributive laws.

Future Work. We plan to build a term assignment for ATLL that can be used a scripting language for defining and reasoning about attack trees. In addition, we plan to extend equivalence of attack trees with contraction for choice, and to investigate adding a modality that adds weakening to ATLL, and then, this modality could be used to reason about subattack trees. Finally, we leave the proof theory of ATLL to future work.

References

1. S.A. Camtepe and B. Yener. Modeling and detection of complex attacks. In *Security and Privacy in Communications Networks and the Workshops, 2007. SecureComm 2007. Third International Conference on*, pages 234–243, Sept 2007.
2. Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not.*, 46(4):53–64, May 2011.
3. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martı-Oliet, José Meseguer, and Carolyn Talcott. Maude manual (version 2.1). *SRI International, Menlo Park*, 2005.
4. Valeria de Paiva. A Dialectica model of the Lambek calculus. In *8th Amsterdam Logic Colloquium*, 1991.
5. Olga Gadyatskaya and Rolando Trujillo-Rasua. New directions in attack tree research: Catching up with industrial needs. In Peng Liu, Sjouke Mauw, and Ketil Stolen, editors, *Graphical Models for Security*, pages 115–126, Cham, 2018. Springer International Publishing.
6. Ross Horne, Sjouke Mauw, and Alwen Tiu. Semantics for specialising attack trees based on linear logic. *Fundamenta Informaticae*, 153(1-2):57–86, 2017.
7. Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer International Publishing, 2015.
8. Simon Peyton Jones. *Haskell 98 language and libraries: the revised report*. Cambridge University Press, 2003.
9. Barbara Kordy, Piotr Kordy, and Yoann van den Boom. *SPTool – Equivalence Checker for* `SAND` *Attack Trees*, pages 105–113. Springer International Publishing, Cham, 2017.
10. Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of attack–defense trees. In Pierpaolo Degano, Sandro Etalle, and Joshua Guttman, editors, *Formal Aspects of Security and Trust*, pages 80–95, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
11. Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational aspects of attack–defense trees. In Pascal Bouvry, MieczysławA. Kłopotek, Franck Leprévost, Małgorzata Marciniak, Agnieszka Mykowiecka, and Henryk Rybiński, editors, *Security and Intelligent Information Systems*, volume 7053 of *Lecture Notes in Computer Science*, pages 103–116. Springer Berlin Heidelberg, 2012.
12. Barbara Kordy, Marc Pouly, and Patrick Schweitzer. A probabilistic framework for security scenarios with dependent actions. In Elvira Albert and Emil Sekerinski, editors, *Integrated Formal Methods*, volume 8739 of *Lecture Notes in Computer Science*, pages 256–271. Springer International Publishing, 2014.

13. Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In DongHo Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin Heidelberg, 2006.
14. J. P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 Workshop on New Security Paradigms*, NSPW '00, pages 15–21, New York, NY, USA, 2000. ACM.
15. Peter O'hearn. On bunched typing. *Journal of functional Programming*, 13(4):747–796, 2003.
16. L. Piètre-Cambacédès and M. Bouissou. Beyond attack trees: Dynamic security modeling with boolean logic driven markov processes (bdmp). In *Dependable Computing Conference (EDCC), 2010 European*, pages 199–208, April 2010.
17. Bruce Schneier. Attack trees: Modeling security threats. *Dr. Dobb's journal*, December 1999.
18. Niki Vazou, Eric L. Seidel, Ranjit Jhala, Dimitrios Vytiniotis, and Simon Peyton-Jones. Refinement types for haskell. *SIGPLAN Not.*, 49(9):269–282, August 2014.